



## REPLACEMENT SHEET

1/5

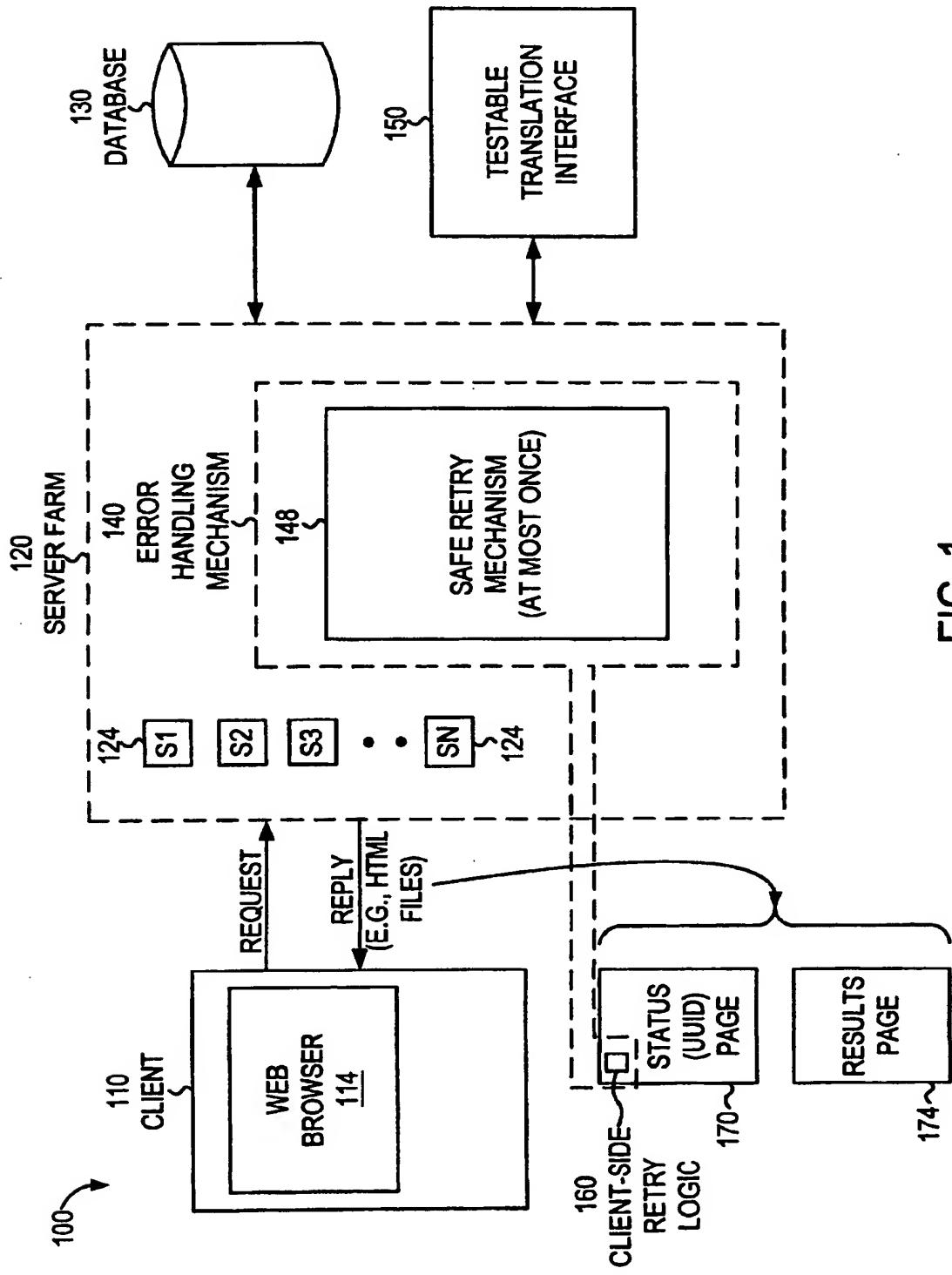


FIG. 1

REPLACEMENT SHEET

2/5

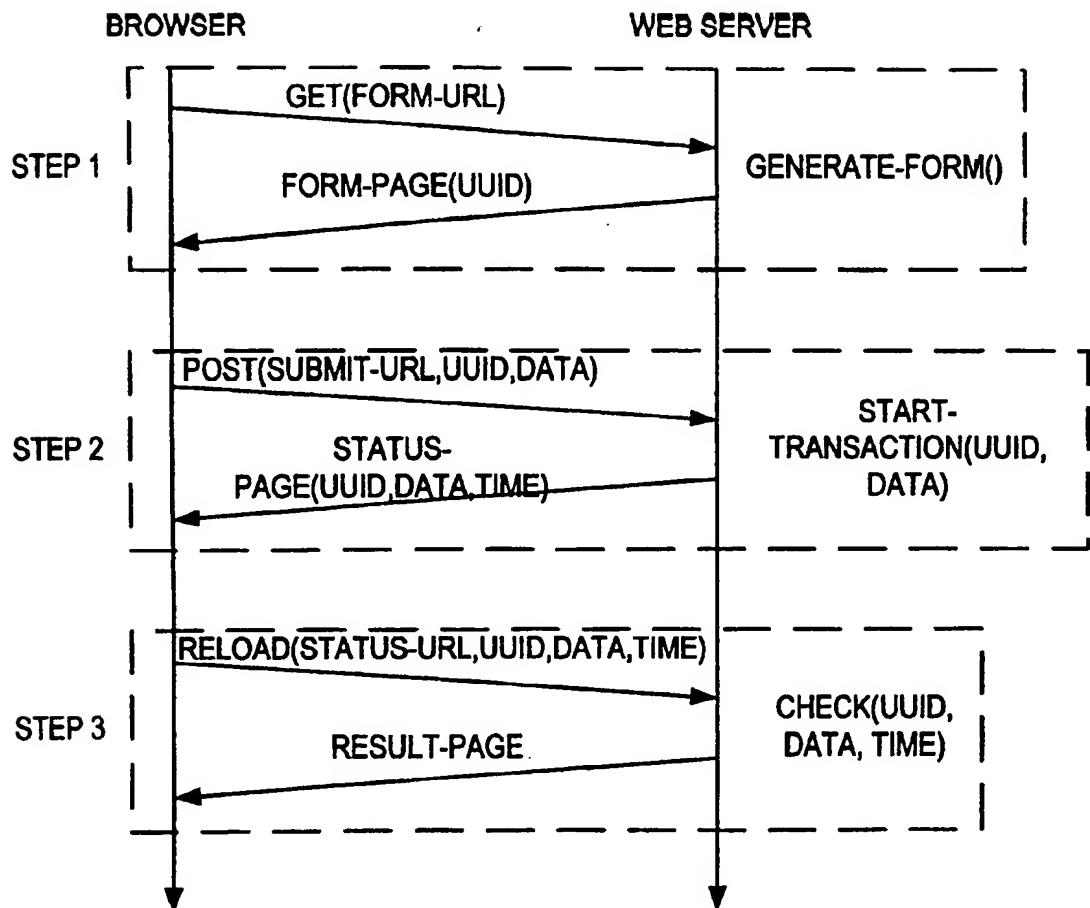


FIG. 2

REPLACEMENT SHEET

3/5

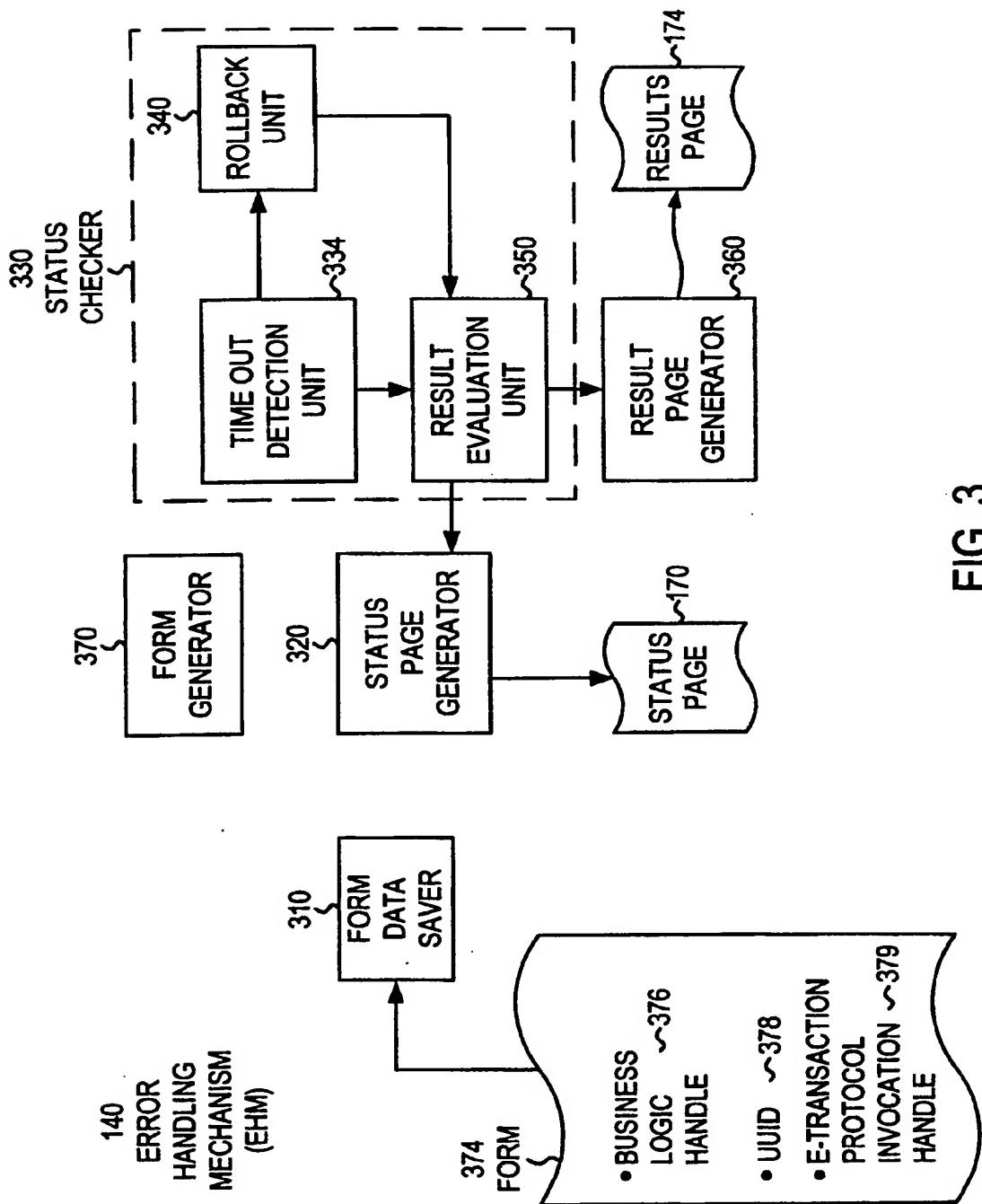


FIG. 3

## REPLACEMENT SHEET

4/5

```
html generate-form() {
    uuid := new UUID();
    return form-html(uuid);
}

void biz-logic(uuid,data) {
    start(uuid);
    // Execute SQL with data as argument.
    // Store result in the variable res
    testable::commit(res,uuid);
}

html start-transaction(uuid,data) {
    // Spawn new thread to execute biz-logic
    // with uuid and data as arguments
    time := current-time();
    return stat-page-html(uuid,data,time);
}

html check(uuid,data,time) {
    if current-time() - time > timeout then
        testable::rollback(uuid);
    res := testable::get-outcome(uuid);
    if res == nil then
        return start-transaction(uuid,data);
    else
        return result-page-html(res);
    else
        res := testable::get-outcome(uuid);
        if res != nil then
            return result-page-html(res);
        else
            return stat-page-html(uuid,data,time);
}
```

FIG. 4

## REPLACEMENT SHEET

5/5

```
void FormServlet(HttpServletRequest req,
                  HttpServletResponse resp) {
    request.getSession(true);
    // Normal Form Creation processing with Form handling servlet name stored
    // in a hidden field, and action set to the Start servlet}
void BizLogic(HttpServletRequest req,
               HttpServletResponse resp) {
    WebDriver.getConnection(`JDBC-URL`);
    // Execute JDBC commands and other logic
    // Output result normally; Do NOT commit any JDBC updates}
void Start(HttpServletRequest req,
            HttpServletResponse resp) {
    WorkQueue.queue(req.clone(),
                    resp.clone());
    SendStatPage(req, resp);}
void Check(HttpServletRequest req,
            HttpServletResponse resp) {
    result = getOutcome(req.getSession());
    if (result != null) {
        result.send(resp);
        return;}
    job = getJobFromQueue(req.getSession());
    if (currentTime() - job.time > timeout) {
        job.abort();
        WorkQueue.queue(job.req, job.resp);
    } else {SendStatPage(job.req, job.resp);}}
void WorkerThread() {
    while(true) {
        job = getJobFromQueue();
        servlet = job.targetServlet;
        try {
            // Begin Transaction
            servlet.service(job.req, job.resp);
            storeOutcome(job.req, job.resp);
            removeJobFromQueue(job);
            // Commit Transaction
        } catch (Exception e) {
            // Abort transaction}}}
```

FIG. 5